Splitify Project

Documentazione generale

Domingo Dirutigliano

Politecnico di Bari
Fondamenti Web

Contents

1 Scenario applicativo

Splitify è un'applicazione web progettata per semplificare la gestione e la divisione delle spese tra gruppi di persone. Questo strumento è particolarmente utile per gestire spese condivise in contesti come viaggi, cene, eventi o convivenze, dove più persone contribuiscono finanziariamente e necessitano di un sistema per tenere traccia di chi ha pagato cosa e come dovrebbero essere suddivise le spese.

1.1 Funzionalità principali

L'applicazione Splitify offre le seguenti funzionalità principali:

- Creazione e gestione di diverse *board* (lavagne) per organizzare le spese per eventi o gruppi specifici
- Aggiunta di membri all'interno di una board, rappresentanti le persone coinvolte nelle spese
- Creazione di categorie per classificare le diverse tipologie di spese
- Registrazione di prodotti con relativi prezzi, associati a specifiche categorie
- Tracciamento delle transazioni finanziarie tra membri del gruppo
- Calcolo automatico dei saldi e delle somme dovute da ciascun membro
- Ottimizzazione dei rimborsi attraverso un algoritmo di backtracking che calcola la sequenza ottimale di transazioni per minimizzare il numero di scambi necessari
- Condivisione delle board con altri utenti con diversi livelli di permessi

1.2 Architettura dell'applicazione

Splitify è sviluppato come applicazione web full-stack seguendo un'architettura client-server:

- **Frontend**: Implementato utilizzando React.js, TypeScript e Mantine UI per l'interfaccia utente. Il frontend comunica con il backend tramite chiamate API REST e connessioni WebSocket per gli aggiornamenti in tempo reale.
- **Backend**: Sviluppato utilizzando Node.js con il runtime Bun e TypeScript, fornisce API RESTful per la gestione dei dati e implementa logiche di business. Utilizza WebSocket per la comunicazione in tempo reale con il frontend.
- **Database**: MongoDB viene utilizzato come database NoSQL per la persistenza dei dati.
- **Autenticazione**: Il sistema utilizza autenticazione basata su JWT (JSON Web Token) per gestire sessioni e autorizzazioni degli utenti.

L'applicazione è containerizzata utilizzando Docker, facilitando il deployment in diversi ambienti e la configurazione del sistema.

1.3 Sistema di autenticazione

Splitify implementa un sistema di autenticazione robusto basato su token JWT (JSON Web Token):

- **Autenticazione iniziale**: Quando un utente effettua il login con le proprie credenziali, il server verifica l'identità dell'utente e, se le credenziali sono valide, genera un JWT firmato contenente l'identificativo dell'utente e della sessione.
- **Struttura del token**: Il token JWT contiene informazioni essenziali nel payload come:
  - `sub`: Identificativo dell'utente
  - `sid`: Identificativo univoco della sessione
  - `exp`: Timestamp di scadenza del token
  - `iat`: Timestamp di emissione del token
- **Gestione delle sessioni**: Il sistema tiene traccia delle sessioni attive dell'utente nel database, memorizzando per ciascuna sessione un identificativo univoco, la data di creazione, la data di scadenza e l'ultimo utilizzo.
- **Meccanismo di refresh**: Per migliorare la sicurezza senza compromettere l'esperienza utente, i token hanno una durata limitata. Quando un token sta per scadere, il client può richiamare l'endpoint `/token/refresh` per ottenere un nuovo token senza richiedere nuovamente le credenziali dell'utente. Questo meccanismo:
  - Verifica che il token attuale sia ancora valido
  - Controlla che la sessione corrispondente sia attiva nel database
  - Aggiorna il timestamp di ultimo utilizzo della sessione
  - Genera un nuovo token con una nuova scadenza estesa

  Il refresh del token può essere effettuato solo nell'ultimo 20% della sua durata totale. Il frontend implementa un sistema intelligente di refresh che, decodificando il token JWT, calcola automaticamente quando è necessario rinnovarlo e richiama l'endpoint di refresh in modo asincrono nell'ultimo 9% della durata del token, evitando così interruzioni dell'esperienza utente o ricaricamenti della pagina.
- **Opzione "mantieni accesso"**: Durante il login, l'utente può scegliere l'opzione "mantieni accesso" (`keepLogin`), che estende significativamente la durata della sessione memorizzata nel database, permettendo refresh del token per periodi più lunghi.
- **Gestione delle sessioni multiple**: Il sistema supporta fino a un massimo di 10 sessioni attive contemporaneamente per ciascun utente. Quando viene raggiunto questo limite e l'utente effettua un nuovo login, la sessione più vecchia viene automaticamente invalidata. L'utente può inoltre invalidare tutte le sessioni attive semplicemente cambiando la propria password.

Ogni richiesta API che richiede autenticazione include il token JWT nell'header di autorizzazione. Un middleware del backend verifica la validità del token prima di permettere l'accesso alle risorse protette.

2 Diagrammi UML

2.1 Diagramma dei casi d'uso

Il diagramma dei casi d'uso illustra le interazioni tra gli attori del sistema (utenti non autenticati, utenti registrati e amministratori) e le funzionalità offerte dall'applicazione.

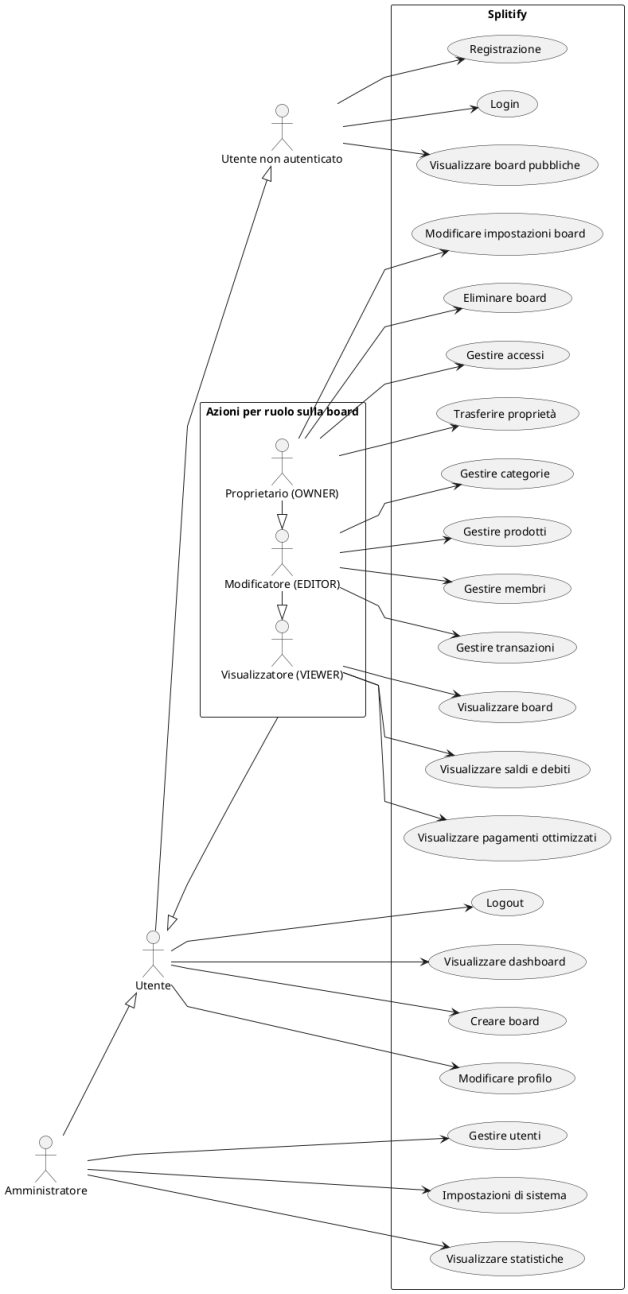

Figure 1: Diagramma dei casi d'uso di Splitify

Gli attori principali sono:

- **Utente non autenticato**: può registrarsi al sistema o effettuare il login. Inoltre, ha la

possibilità di visualizzare le board che sono state impostate come pubbliche, anche senza autenticazione.

- **Utente**: dopo l'autenticazione, può gestire board, categorie, prodotti, membri e transazioni, oltre a poter modificare il proprio profilo.
- **Amministratore**: oltre a tutte le funzionalità dell'utente, ha accesso alle funzionalità di amministrazione come la gestione degli utenti, le impostazioni di sistema e la visualizzazione delle statistiche.

2.2 Modello dei dati

Il seguente diagramma mostra il modello dei dati utilizzato in Splitify, rappresentando le entità principali e le loro relazioni:

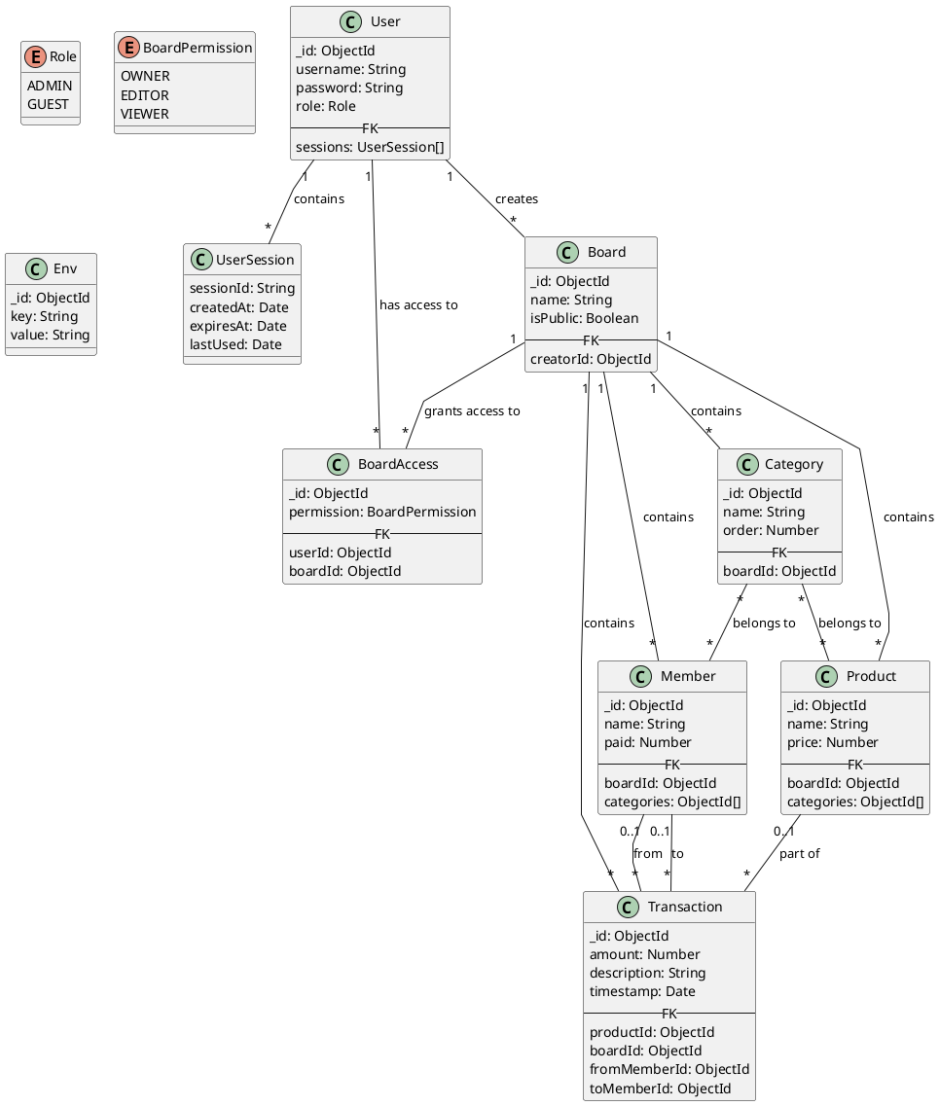

Figure 2: Modello dei dati di Splitify

3 Modello dei dati

Nel diagramma ER sono state volutamente inserite le chiavi esterne anche se normalmente, nella rappresentazione Entity-Relationship pura, queste non andrebbero incluse. Questa scelta è stata fatta per evidenziare esplicitamente come sono state implementate le associazioni tra le entità in MongoDB, un database NoSQL. Il modello dei dati non segue la normalizzazione tipica dei database relazionali, ma è strutturato secondo il paradigma document-oriented di MongoDB, dove i riferimenti tra documenti vengono implementati tramite campi che contengono gli identificatori (ObjectId) di altri documenti. Il modello dei dati di Splitify è composto dalle seguenti entità principali:

3.1 User

Rappresenta un utente registrato nel sistema.

- **_id**: Identificatore univoco (ObjectId)
- **username**: Nome utente univoco
- **password**: Password hashata
- **role**: Ruolo dell'utente (ADMIN o GUEST)
- **sessions**: Array di sessioni utente attive

3.2 UserSession

Rappresenta una sessione di autenticazione attiva.

- **sessionId**: Identificatore univoco della sessione
- **createdAt**: Data e ora di creazione
- **expiresAt**: Data e ora di scadenza
- **lastUsed**: Data e ora dell'ultimo utilizzo

3.3 Board

Rappresenta una lavagna di gestione spese.

- **_id**: Identificatore univoco (ObjectId)
- **name**: Nome della board
- **isPublic**: Flag che indica se la board è pubblica
- **creatorId**: Riferimento all'utente creatore (ObjectId)

3.4 BoardAccess

Gestisce i permessi di accesso alle board per gli utenti.

- **_id**: Identificatore univoco (ObjectId)
- **userId**: Riferimento all'utente (ObjectId)
- **boardId**: Riferimento alla board (ObjectId)
- **permission**: Livello di permesso (OWNER, EDITOR, VIEWER)

3.5 Category

Rappresenta una categoria di spese.

- **_id**: Identificatore univoco (ObjectId)
- **boardId**: Riferimento alla board (ObjectId)
- **name**: Nome della categoria
- **order**: Ordine di visualizzazione

3.6 Member

Rappresenta un membro partecipante alla divisione delle spese.

- **_id**: Identificatore univoco (ObjectId)
- **boardId**: Riferimento alla board (ObjectId)
- **name**: Nome del membro
- **paid**: Importo pagato
- **categories**: Array di riferimenti a categorie (ObjectId[])

3.7 Product

Rappresenta un prodotto o servizio acquistato.

- **_id**: Identificatore univoco (ObjectId)
- **boardId**: Riferimento alla board (ObjectId)
- **name**: Nome del prodotto
- **price**: Prezzo del prodotto
- **categories**: Array di riferimenti a categorie (ObjectId[])

3.8 Transaction

Rappresenta un movimento finanziario tra membri.

- **_id**: Identificatore univoco (ObjectId)
- **boardId**: Riferimento alla board (ObjectId)
- **fromMemberId**: Riferimento al membro pagante (ObjectId, opzionale)
- **toMemberId**: Riferimento al membro beneficiario (ObjectId, opzionale)
- **amount**: Importo della transazione
- **description**: Descrizione della transazione
- **productId**: Riferimento al prodotto associato (ObjectId, opzionale)
- **timestamp**: Data e ora della transazione

3.9 Env

Memorizza variabili di configurazione dell'ambiente.

- **_id**: Identificatore univoco (ObjectId)
- **key**: Nome della variabile
- **value**: Valore della variabile

4 API Backend

Il backend di Splitify espone una serie di API RESTful per interagire con il sistema. Le API sono organizzate nei seguenti gruppi:

4.1 Autenticazione (/)

- **POST /login**: Autentica un utente e restituisce un token JWT. Accetta i parametri `username`, `password` e un flag opzionale `keepLogin` per mantenere la sessione attiva più a lungo.
- **GET /me**: Restituisce i dati dell'utente autenticato basandosi sul token JWT fornito nell'header di autorizzazione.
- **POST /register**: Registra un nuovo utente con username e password. La registrazione può essere aperta a tutti o richiedere un token, a seconda delle impostazioni del sistema.
- **POST /register/:token**: Registra un nuovo utente utilizzando un token di invito specifico, utile quando la registrazione è ristretta.
- **POST /token/refresh**: Rinnova il token di autenticazione generando un nuovo token JWT con una nuova data di scadenza, mantenendo la stessa sessione dell'utente.
- **GET /register/info**: Ottiene informazioni sulla modalità di registrazione attualmente configurata.
- **PUT /register/set**: Imposta la modalità di registrazione del sistema (aperta, chiusa o con token). Riservata agli amministratori.

4.2 Gestione Board (/boards)

- **GET /boards**: Ottiene la lista delle board dell'utente
- **GET /boards/:id**: Ottiene i dettagli di una specifica board
- **POST /boards**: Crea una nuova board
- **PUT /boards/:id**: Aggiorna una board esistente
- **DELETE /boards/:id**: Elimina una board
- **GET /boards/:id/access**: Ottiene la lista degli accessi a una board
- **POST /boards/:id/access**: Concede accesso a un utente
- **PUT /boards/:id/access/:userId**: Modifica i permessi di un utente
- **DELETE /boards/:id/access/:userId**: Revoca l'accesso di un utente
- **PUT /boards/:id/transfer**: Trasferisce la proprietà della board
- **GET /boards/:id/categories**: Ottiene le categorie di una board
- **POST /boards/:id/categories**: Aggiunge una categoria a una board
- **PUT /boards/:id/categories/:category_id**: Aggiorna una categoria
- **DELETE /boards/:id/categories/:category_id**: Elimina una categoria
- **GET /boards/:id/members**: Ottiene i membri di una board
- **POST /boards/:id/members**: Aggiunge un membro a una board
- **PUT /boards/:id/members/:member_id**: Aggiorna un membro
- **DELETE /boards/:id/members/:member_id**: Elimina un membro
- **GET /boards/:id/products**: Ottiene i prodotti di una board
- **POST /boards/:id/products**: Aggiunge un prodotto a una board
- **PUT /boards/:id/products/:product_id**: Aggiorna un prodotto
- **DELETE /boards/:id/products/:product_id**: Elimina un prodotto

4.3 Gestione Transazioni (/transactions)

- **GET /transactions/:boardId**: Ottiene le transazioni di una board
- **POST /transactions/:boardId**: Crea una nuova transazione

4.4 Gestione Utenti (/users)

- **GET /users/utils/search**: Cerca utenti nel sistema
- **PUT /users/me/username**: Aggiorna il nome utente
- **DELETE /users**: Elimina l'account dell'utente corrente
- **PUT /users/me/password**: Cambia la password dell'utente corrente
- **GET /users**: Ottiene la lista degli utenti (solo admin)
- **GET /users/:id**: Ottiene i dettagli di un utente specifico (solo admin)
- **POST /users**: Crea un nuovo utente (solo admin)
- **PUT /users/:id**: Aggiorna un utente (solo admin)
- **DELETE /users/:id**: Elimina un utente (solo admin)

4.5 Amministrazione (/admin)

- **GET /admin/stats**: Ottiene statistiche del sistema (solo admin)

5 Sistema di permessi

Splitify implementa un sistema granulare di controllo degli accessi che consente la collaborazione tra utenti con diversi livelli di permesso sulle board. Ogni board può essere condivisa con altri utenti, assegnando loro uno dei seguenti livelli di permesso:

- **OWNER (Proprietario):** Rappresenta il livello di permesso più elevato.
  - Ha accesso completo a tutte le funzionalità della board.
  - Può visualizzare e modificare tutti i dati (membri, categorie, prodotti, transazioni).
  - Può modificare le impostazioni della board (nome, visibilità).
  - Può gestire gli accessi degli altri utenti (aggiunta, modifica, rimozione).
  - Può trasferire la proprietà della board ad un altro utente.
  - Può eliminare definitivamente la board.
  - Solo il creatore originale della board o un utente a cui è stata trasferita la proprietà può avere questo permesso.
- **EDITOR (Modificatore):** Rappresenta un livello intermedio di permesso.
  - Può visualizzare tutti i dati della board.
  - Può aggiungere, modificare ed eliminare membri, categorie, prodotti e transazioni.
  - Può utilizzare la funzionalità di ottimizzazione dei pagamenti.
  - Non può modificare le impostazioni della board (nome, visibilità).
  - Non può gestire gli accessi degli altri utenti.
  - Non può eliminare la board.

  - Questo livello è assegnato automaticamente al proprietario precedente quando trasferisce la proprietà della board.

- **VIEWER (Visualizzatore):** Rappresenta il livello più basso di permesso.
  - Può solo visualizzare tutti i dati della board (membri, categorie, prodotti, transazioni).
  - Può consultare i riepiloghi dei saldi e debiti.
  - Può vedere l'elenco ottimizzato dei pagamenti consigliati.
  - Non può effettuare alcuna modifica ai dati.
  - Non può registrare transazioni o aggiornare lo stato dei pagamenti.

  Si noti come se impostiamo una board come pubblica, qualunque utente, anche non registrato e non autenticato, può visualizzare i dati della board, e avrà i medesimi permessi di un utente VIEWER. Ciò è possibile conoscendo il link della board. (Il link della board non sarà accessibile se la board è privata da utenti non autorizzati)

5.1 Implementazione tecnica

Il sistema di permessi è implementato attraverso il modello `BoardAccess`, che associa un utente a una board con un livello specifico di permesso. Quando un utente accede a una board, il backend verifica il livello di permesso associato e adatta di conseguenza le operazioni consentite. Ad esempio:

- Le richieste di modifica vengono respinte se provenienti da utenti con permesso VIEWER.
- Le operazioni di gestione della board (modifica delle impostazioni, gestione degli accessi) sono consentite solo a utenti con permesso OWNER.
- Le operazioni di modifica dei dati (aggiunta di membri, registrazione di transazioni) sono consentite a utenti con permesso EDITOR o superiore.

Il frontend adatta l'interfaccia utente in base al livello di permesso, mostrando o nascondendo elementi interattivi come pulsanti di modifica, eliminazione o impostazioni, e visualizzando indicatori che informano l'utente del suo livello di accesso alla board.

6 Componenti Frontend

Il frontend di Splitify è sviluppato utilizzando React con TypeScript e la libreria di componenti Mantine UI. L'architettura frontend segue un approccio modulare con componenti specializzati per ciascuna funzionalità dell'applicazione.

6.1 Struttura e organizzazione

Il codice frontend è organizzato in diverse cartelle principali:

- **src/commons/**: Componenti riutilizzabili in diverse parti dell'applicazione
- **src/components/**: Componenti specifici per le diverse sezioni dell'applicazione
- **src/styles/**: File CSS e stili condivisi
- **src/utils/**: Utility, hook personalizzati e funzioni di supporto
- **src/workers/**: Web worker per operazioni computazionali intensive

6.2 Layout e navigazione

- **App.tsx**: Componente root dell'applicazione che definisce la struttura principale dell'interfaccia utente. Gestisce:
  - Setup del routing tramite React Router
  - Inizializzazione delle connessioni WebSocket
  - Caricamento lazy dei componenti per ottimizzare le performance
  - Gestione degli stati globali dell'applicazione
  - Struttura del layout responsive tramite AppShell di Mantine
- **NavigatorContext**: Implementa tutti i componenti base necessari ad attivare la navigazione di react-router, ma permettendo di utilizzare le funzionalità di navigazione anche in componenti esterni a questo: Il componente è impropriamente chiamato "Context" ma non è un reale ContextProvider, invece questo fa utilizzo degli state globali di zustand per permettere l'utilizzo di useNavigate nei componenti di AppShell di Mantine che sono esterni ai componenti di routing.
- **BurgerSection**: Menu laterale adattivo che è visibile solo su dispositivi mobili e si comporta come menu a scomparsa. Su desktop, il menu è integrato nel layout come sidebar fissa e contiene i link di navigazione principali dell'applicazione.
- **AdminLayout**: Layout specializzato per la sezione amministrativa con:
  - Menu di navigazione specifico per le funzionalità admin
  - Controlli di accesso avanzati basati sul ruolo

6.3 Autenticazione e gestione account

- **LoginProvider**: Gestisce il processo di autenticazione con:
  - Form di login con validazione degli input
  - Gestione degli errori di autenticazione
  - Memorizzazione sicura del token JWT con refresh automatico
  - Supporto per l'opzione "mantieni accesso"
- **RegisterForm** e **RegisterPage**: Implementano il processo di registrazione con:
  - Validazione dei dati in tempo reale
  - Gestione dei diversi modi di registrazione (aperta, con token)
- **UserProfilePage**: Centro di controllo del profilo utente che permette di:
  - Modificare le informazioni del profilo
  - Cambiare la password con verifica della password attuale
  - Visualizzare e terminare le sessioni attive
  - Eliminare l'account con conferma di sicurezza
- **UserInfoDisplay**: Componente visuale che:
  - Mostra le informazioni dell'utente corrente
  - Fornisce accesso rapido alle funzioni di gestione account

6.4 Dashboard e gestione board

- **Dashboard**: Pagina principale post-login che:
  - Visualizza le board dell'utente con layout a griglia o lista
  - Permette la creazione di nuove board
  - Mostra board condivise
  - Implementa funzionalità di ricerca e filtro
  - Fornisce indicatori visivi sui permessi di accesso alle board
- **BoardPage**: Hub centrale per la gestione di una specifica board:
  - Orchestrazione dei componenti figli (categorie, membri, prodotti, transazioni)
  - Gestione del sistema di tab per navigare tra le diverse sezioni (tabella membri e tabella prodotti)
  - Visualizzazione e modifica dei dettagli e permessi sulla board
  - Calcolo e visualizzazione dei saldi e dei debiti tra membri

  Si specifica che il calcolo dei debiti, dei saldi e pagamenti consigliati è eseguito direttamente nel frontend: il backend non è coinvolto in questo processo. Questo permette di ridurre nettamente il carico sul backend e di semplificare la gestione di questi dati, con l'accortezza di elaborarli coerente e deterministicamente su tutti i dispositivi su cui andiamo a visualizzarli, evitando disallineamenti e comportamenti inconsistenti.
- **AddBoardModal**: Interfaccia per la creazione di nuove board con:
  - Form di inserimento con validazione
  - Opzione per impostare la visibilità pubblica/privata
- **BoardAccessModal**: Gestione avanzata degli accessi con:
  - Ricerca di utenti tramite username (tramite apposita API che permette di conoscere informazioni limitate sugli utenti)
  - Assegnazione di permessi specifici (editor, viewer)
  - Revoca degli accessi esistenti
  - Trasferimento di proprietà della board

6.5 Gestione di categorie, membri, prodotti e transazioni

- **Componenti per categorie**:
  - **CategorySettingsModal**: Componente principale per la gestione delle categorie che consente la visualizzazione, la modifica e l'eliminazione di categorie esistenti. Include una tabella paginata e permette di riordinare le categorie tramite drag-and-drop.
  - **CategoryRow**: Componente che rappresenta una singola riga nella tabella delle categorie, con campi modificabili in-line per il nome e controlli per l'ordinamento e l'eliminazione.
  - **AddCategoryModal**: Modal per l'inserimento di nuove categorie con validazione del nome.

- **DeleteCategory**: Componente di conferma per l'eliminazione sicura di una categoria, che mostra avvisi nel caso in cui la categoria sia utilizzata da membri o prodotti.

- **Componenti per membri**:
  - **MemberSettingsModal**: Componente principale per la gestione dei membri, che adatta la sua visualizzazione in base alla dimensione dello schermo. Permette di visualizzare, modificare e rimuovere membri con relativi importi pagati e categorie associate.
  - **MemberTableRow**: Visualizzazione tabellare di un singolo membro con campi modificabili per nome, importo pagato e categorie. Include anche indicatori visuali per i saldi (credito, debito o in pareggio).
  - **MemberMobileCard**: Versione alternativa del MemberTableRow specifica per dispositivi mobili, con layout ottimizzato per schermi di dimensioni ridotte.
  - **AddMemberModal**: Interfaccia per l'aggiunta di nuovi membri con validazione dei dati inseriti.
  - **DeleteMember**: Componente di conferma per l'eliminazione di un membro, con avvisi relativi alle transazioni associate.
  - **NoMembersPlaceholder**: Componente visuale mostrato quando una board non ha ancora membri, con indicazioni per l'utente su come procedere.

- **Componenti per prodotti**:
  - **ProductSettingsModal**: Componente principale per la gestione dei prodotti con visualizzazione paginata e responsiva. Consente di visualizzare, filtrare, modificare ed eliminare prodotti.
  - **ProductRow**: Rappresentazione tabellare di un singolo prodotto con campi modificabili per nome, prezzo e categorie associate.
  - **AddProductModal**: Interfaccia per l'aggiunta di nuovi prodotti con validazione del nome e del prezzo. Permette la selezione multipla di categorie a cui associare il prodotto.
  - **DeleteProduct**: Componente di conferma per l'eliminazione di un prodotto, che verifica e avvisa se il prodotto è associato a transazioni.

- **Componenti per transazioni e pagamenti**:
  - **TransactionsModal**: Interfaccia completa per la visualizzazione e gestione delle transazioni, organizzata come una timeline cronologica con paginazione.
  - **PaymentListModal**: Componente avanzato che visualizza l'elenco ottimizzato dei pagamenti consigliati calcolati dal sistema per saldare i debiti tra membri.
  - **PaymentItem**: Elemento visuale che rappresenta un singolo pagamento consigliato, con informazioni sul mittente, destinatario e importo.
  - **ConfirmationPaymentModalContent**: Interfaccia di conferma per registrare un pagamento consigliato come transazione effettiva, mostrando chiaramente le conseguenze dell'operazione.

6.6 Amministrazione

- **AdminDashboard**: Centro di controllo per gli amministratori che include:
  - Panoramica dello stato del sistema
  - Metriche di utilizzo (utenti attivi, board create, transazioni)
  - Accesso rapido alle funzionalità amministrative
  - Notifiche di sistema e avvisi
- **UserManagementPage**: Gestione completa degli utenti con:
  - Lista utenti
  - Creazione di nuovi utenti con assegnazione di ruoli
  - Modifica dei dati utente e reset password
  - Eliminazione account
  - Visualizzazione ultimo accesso
- **SystemSettings**: Configurazione del sistema con:
  - Impostazioni della modalità di registrazione
  - Future impostazioni di sistema

6.7 Componenti comuni e riutilizzabili

L'applicazione include una ricca libreria di componenti riutilizzabili:

- **Buttons**: Collezione di pulsanti specializzati per diverse azioni:
  - **HomeButton**: Per tornare alla dashboard
  - **AdminButton**: Per accedere alla sezione admin
  - **LogoutButton**: Per disconnettersi
  - **EditButton**, **DeleteButton**, **SaveButton**: Per operazioni CRUD
- **YesOrNoModal**: Dialogo di conferma personalizzabile per azioni critiche
- **ModalPaper**: Contenitore standard per i modal
- **FormButtonBox**: Layout standardizzato per posizionare i pulsanti nei form responsive.
- **CategoryCheckbox**: Componente per la selezione multipla di categorie con grafica adattata per le tabelle dei membri e dei prodotti
- **AdvancedNumberInput**: Input numerico avanzato per numeri con la virgola per la memorizzazione di questi con Big.js (evitando problematiche comuni con i float di javascript). Possiede:
  - Validazione numerica
  - Formattazione automatica
- **ResponsivePager**: Sistema di paginazione che si adatta a diverse dimensioni dello schermo

- **PermissionIcon**: Indicatori visivi per i livelli di permesso
- **BalanceIcon**: Visualizzazione grafica dello stato dei saldi
- **EditHeader**: Intestazione standardizzata per le pagine di modifica
- **BottomEditControl**: Barra di controllo fissa in fondo alla pagina per le azioni di salvataggio

6.8 Comunicazione con il backend e gestione dello stato

Il frontend comunica con il backend attraverso diversi meccanismi:

- **API REST**: Implementate tramite il modulo **net.ts** che:
  - Fornisce wrapper per chiamate HTTP
  - Gestisce automaticamente gli header di autenticazione
  - Implementa la gestione centralizzata degli errori
  - Supporta la riconnessione e il retry in caso di fallimento
- **React Query**: Utilizzato in **queries.ts** per:
  - Caching dei dati e riduzione delle chiamate API
  - Gestione dello stato di caricamento e di errore
  - Invalidazione intelligente della cache
  - Aggiornamenti ottimistici dell'interfaccia
- **WebSocket**: Implementato in **socket.ts** per:
  - Aggiornamenti in tempo reale quando altri utenti modificano i dati
  - Notifiche di sistema
  - Riconnessione automatica in caso di interruzione
- **Zustand**: Utilizzato in **store.ts** per:
  - Gestione globale dello stato dell'applicazione
  - Condivisione dello stato tra componenti non correlati
  - Persistenza selettiva dei dati
- **Web Workers**: Implementati per:
  - Calcoli complessi sui saldi ottimali tra membri
  - Algoritmo di backtracking per l'ottimizzazione delle transazioni di rimborso
  - Elaborazione dell'albero decisionale per determinare la catena di pagamenti ottimale
  - Elaborazioni di dati che potrebbero rallentare l'interfaccia utente

6.9 Comunicazione in tempo reale e collaborazione

L'applicazione Splitify implementa un sistema avanzato di comunicazione in tempo reale basato su WebSocket che permette agli utenti di collaborare simultaneamente sulla stessa board. Questo sistema garantisce che tutti gli utenti connessi alla stessa board visualizzino sempre dati aggiornati, creando un'esperienza di editing collaborativo fluida e reattiva.

- **Implementazione WebSocket**:
  - Il backend utilizza Socket.IO per creare canali di comunicazione bidirezionali persistenti tra server e client
  - Ogni socket è autenticato utilizzando lo stesso sistema JWT dell'API REST, garantendo la sicurezza delle connessioni
  - Gli utenti che accedono a una board si "iscrivono" a una stanza (room) specifica per quella board utilizzando il metodo `joinBoardRoom`
  - Le socket sono automaticamente raggruppate in canali basati sul ruolo dell'utente (admin, utente normale) e sulle board a cui hanno accesso
- **Sistema di notifiche e aggiornamenti**:
  - Quando un utente apporta modifiche a una board (aggiunge un membro, modifica un prodotto, ecc.), il backend emette eventi specifici utilizzando funzioni come `emitBoardUpdate`
  - Il server invia ai client connessi alla stessa board un messaggio contenente le `queryKeys` da invalidare
  - Questo messaggio specifica esattamente quali parti dell'interfaccia utente devono essere aggiornate (ad esempio: membri, categorie, prodotti)
- **Invalidazione intelligente della cache React Query**:
  - Quando un client riceve un evento di aggiornamento, React Query invalida selettivamente le chiavi indicate nel messaggio
  - Questo approccio è più efficiente rispetto al ricaricamento completo della pagina o al polling periodico
  - Solo le query specifiche vengono ri-eseguite, riducendo il carico sul server e la quantità di dati trasferiti
- **Esperienza di collaborazione in tempo reale**:
  - Gli utenti vedono immediatamente le modifiche effettuate da altri senza necessità di aggiornare manualmente la pagina
  - Il sistema gestisce automaticamente la riconnessione in caso di interruzioni di rete temporanee

Questo meccanismo permette esperienze collaborative come:

- Più utenti che aggiungono contemporaneamente membri o prodotti alla stessa board
- Visualizzazione in tempo reale dei saldi aggiornati quando vengono registrate nuove transazioni

L'implementazione tecnica di questo sistema si basa su tre componenti principali:

- **socket.ts** nel backend: Gestisce le connessioni WebSocket, l'autenticazione delle socket e l'emissione degli eventi di aggiornamento
- **socket.ts** nel frontend: Si occupa della connessione al server, dell'iscrizione alle stanze delle board e della gestione degli eventi ricevuti
- Listener degli eventi in **App.tsx**: Intercetta gli eventi di aggiornamento e utilizza il metodo `invalidateQueries` di React Query per aggiornare i dati nell'interfaccia